FCModeler User's Manual

Version 1.0

September 2002

Written by

Zach Cox

Julie Dickerson

Adam Tomjack

Copyright Julie Dickerson, Iowa State University 2002

Table of Contents

1 Introduction to FCModeler ... 1
2 Setting Up FCModeler ... 1
2.1 Setting up the FCModelerConfig File ... 1
2.2 Running FCModeler ... 1
3 Sources of Input ... 1
3.1 Graph XML Files ... 1
3.1.1 XML Format ... 1
3.1.2 Example of a Complete XML File ... 5
3.1.3 Opening a Graph XML File ... 6
3.1.4 Saving a Graph XML File ... 7
3.1.5 Saving a JPEG Image of the Graph ... 8
3.2 MySQL Database ... 8
4 Graph Layout ... 8
4.1 Simple Dot Layout ... 8
4.2 Rank-Cluster Dot Layout ... 9
4.2.1 Layout XML File ... 9
4.3 GEM Layout(s) ... 10
5 Interacting with the Graph View ... 12
5.1 Selecting Node and Edge Figures ... 12
5.2 Node Figures ... 13
5.2.1 Adding ... 13
5.2.2 Deleting ... 13
5.2.3 Moving ... 13
5.3 Edge Figures ... 13
5.3.1 Adding ... 14
5.3.2 Modifying ... 14
5.3.3 Deleting ... 14
5.3.4 Moving ... 14
5.4 Zooming ... 15
5.5 Panning ... 15
5.6 Finding a Particular Node ... 15
6 Mappings ... 15
6.1 Node and Edge Properties ... 15
6.1.1 Property Viewer ... 16
6.2 Visual Attributes of Node and Edge Figures ... 17
6.3 Using Visual Attributes to Visualize Properties ... 17
6.4 Creating Mappings Using the Mapping Editor ... 18
7 Animation ... 19
7.1 XML File Format ... 19
7.2 Controls ... 21
8 Graph Theoretic Operations ... 21
8.1.1 Subgraph Creation ... 21
8.1.2 Strongly Connected Components ... 22

8.1.3 Cycle Search 23
8.1.4 Cycle XML Files........ 24
8.2 Alternate Paths between Nodes 25
8.3 Clustering Cycles........ 25
9 FCModeler and R........ 28
10 References........ 28
10.1 Publications relating to FCModeler........ 28
10.2 Open Source Code Used in FCModeler........ 28

List of Figures

Figure 1. The open graph dialog box........ 7
Figure 2. The save graph dialog box........ 7
Figure 3. Simple dot layout........ 8
Figure 4. Rank-cluster dot layout........ 9
Figure 5: Example of a GEM layout........ 11
Figure 6 shows the manipulation of node and edge objects in FCModeler........ 14
Figure 7 shows how to save graph coordinates using the save function under the File menu. 15
Figure 8. Property viewer showing property values of selected nodes and edges........ 17
Figure 9. Mapping editor. 18
Figure 10. Animation control........ 21
Figure 11. The subgraph creation dialog box. 22
Figure 12 Example of selecting cycles and strongly connected components as subgraphs. 23
Figure 13. The cycle search dialog box. 24
Figure 14. Map view showing results of the SOM algorithm. Each graph shows the model of the corresponding map unit, which is the generalized median of the cycles assigned to that map unit. 26
Figure 15. Full view of the bottom-left map unit, showing ten cycles clustered together........ 27

1 Introduction to FCModeler

FCModeler models and visualizes metabolic networks as graphs. Nodes of the graph represent specific biochemicals such as proteins, RNA, and small molecules, or stimuli, such as light, heat, or nutrients. Edges of the graph capture regulatory and metabolic relationships found in biological systems. FCModeler can dynamically display user-specified graphs and animate the results of different modeling algorithms on the graph.

The fuzzy cognitive map modeling software is currently written in Matlab and interfaces with FCModeler via xml graph files. We plan to translate this software at a later date.

2 Setting Up FCModeler

The FCModeler distribution is compatible with Windows 2000/XP and Linux machines. Download the file and unzip into your directory. This version is already compiled and the Java source code is included. If you make any improvements to the code, please send the updated modules back for inclusion in the next distribution. FCModeler requires the installation of Java 1.3.x. It does not currently work with Java 1.4.

2.1 Setting up the FCModelerConfig File

This file contains the path information for finding the *dot* layout program and for selecting the directory of graph information. These paths need to be set correctly to find the layout information and graphs. If your version of FCModeler is stored on the C drive in a folder called *FCModeler*, the file should contain the following lines:

```
pathToDot=c:\fcmodeleruser\dot
pathToGraphs=c:\fcmodeleruser\graphs
```

2.2 Running FCModeler

FCModeler can be run by double-clicking on the run.bat file in your directory or directly from the command line.

3 Sources of Input

FCModeler can currently obtain graphs from two different external sources: XML files and a MySQL database.

3.1 Graph XML Files

FCModeler can read in graphs from specially formatted XML files.

3.1.1 XML Format

The XML files are standalone – that means there is no separate DTD file needed. The DTD information is given at the beginning of the XML file. Additionally, there are three types of information that can be stored in an XML file: the topology of the graph, the mappings between node and edge properties and node and edge figure visual attributes, and coordinates of the node and edge figures. Any combination of the three can be present in an XML file as long as the

corresponding DTD is present. FCModeler allows the user to choose which of the three is actually read from a given XML file.

3.1.1.1 Topology

The topology specifies the nodes and edges of the graph. Each node and edge must have a unique id that is just some string of characters. The nodes and edges can also have multiple properties and each individual node and edge has a specific value for each property. The following shows an example of the XML topology:

```
<graph>
	<nodeProperties>"type" "location" </nodeProperties>
	<node id="u-1">"dna" "loc1" </node>
	<node id="v">"rna" "loc1" </node>
	<node id="w">"molecule" "loc2" </node>
	<node id="z">"molecule" "loc1" </node>
	<node id="a">"enzyme" "loc2" </node>
	<node id="b">"enzyme" "loc1" </node>
	<edgeProperties>"type" "strength" </edgeProperties>
	<edge id="e1" tail="u-1" head="v" directed="true">"regulate" "1.0" </edge>
	<edge id="e3" tail="u-1" head="w" directed="true">"regulate" "1.0" </edge>
	<edge id="e7" tail="u-1" head="a" directed="true">"convert" "-1.0" </edge>
	<edge id="e2" tail="v" head="u-1" directed="true">"regulate" "1.0" </edge>
	<edge id="e4" tail="w" head="v" directed="true">"regulate" "-1.0" </edge>
	<edge id="e5" tail="w" head="z" directed="true">"catalyze" "1.0" </edge>
	<edge id="e6" tail="z" head="w" directed="true">"convert" "-1.0" </edge>
	<edge id="e8" tail="a" head="b" directed="true">"convert" "-1.0" </edge>
	<edge id="e9" tail="b" head="v" directed="true">"convert" "-1.0" </edge>
</graph>
```

The `<graph>` tag surrounds the topology section. The `<nodeProperties>` and `<edgeProperties>` tags specify the node and edge properties, respectively. Each string surrounded by double-quotes is defined as a single node or edge property.

A single `<node>` tag specifies each node. The unique node ID is given by the `id` element. The text between the beginning and end node tags contains the values for the node properties. Each string surrounded by double-quotes is the value of a property, in the order defined by the `<nodeProperties>` tag.

A single `<edge>` tag specifies each edge. The unique edge ID is given by the `id` element. The `tail` and `head` elements specify the tail and head nodes of the edge, respectively. The `directed` element indicates whether the edge is directed. Like the node tags, the text for the edge tags specifies the values of the edge properties for the particular edge.

The DTD for the topology XML is given by:

```
<!ELEMENT graph (nodeProperties*, node*, compositeNode*, edgeProperties*, edge*)>
<!ELEMENT nodeProperties (#PCDATA)>
<!ELEMENT node (#PCDATA)>
<!ATTLIST node
id ID #REQUIRED>
<!ELEMENT edgeProperties (#PCDATA)>
<!ELEMENT edge (#PCDATA)>
<!ATTLIST edge
id ID #REQUIRED
head IDREF #REQUIRED
tail IDREF #REQUIRED
directed (true|false) "false">
```

This DTD must be included at the beginning of the XML file if the file contains topology information.

3.1.1.2 Mappings

Node and edge properties can be mapped to the visual attributes of node and edge figures. These mappings can be defined in an XML file. An example with three different mappings is given below, based on the graph specified in the Topology section above.

```
<mappings>
        <mapping type="node">
            <atom property="location" value="loc2"/>
            <attributeValue attribute="fill" value="[0.2,1.0,0.4]"/>
        </mapping>
        <mapping type="edge">
            <atom property="strength" value="-1.0"/>
            <attributeValue attribute="color" value="[1.0,0.2,0.0]"/>
        </mapping>
        <mapping type="node">
            <composite>
                <composite>
                    <atom property="location" value="loc1"/>
                    <connective type="and"/>
                    <atom property="type" value="molecule"/>
                </composite>
                <connective type="or"/>
                <atom property="type" value="enzyme"/>
            </composite>
            <attributeValue attribute="node shape" value="ellipse"/>
        </mapping>
</mappings>
```

The `<mappings>` tag surrounds all of the mappings in the XML file. Each individual mapping is specified by a `<mapping>` tag, with the `type` attribute specifying if it is a node or an edge mapping.

Each mapping represents a statement of the form, “If location equals loc2 then fill equals the RGB color value [0.2, 1.0, 0.4].” More complex statements like, “If either location equals loc1 and type equals molecule or type equals enzyme then shape equals ellipse” can also be defined. Therefore, each mapping consists of two parts: the specification of property value(s) and specification of a visual attribute. The `<attributeValue>` tag specifies the visual attribute and value of the mapping.

The `<atom>` tag represents an individual property value; the `property` and `value` attributes provide the details. The first example in the preceding paragraph is shown above in XML as the first mapping.

The `<composite>` tag is used to join two property values together by an AND or an OR logical connective, specified by the `<connective>` tag. Composites can also be nested inside each other to create complex mapping statements. The second example in the preceding paragraph is shown above in XML format in the third `<mapping>` tag.

The DTD for the topology XML is given by:

```
<!ELEMENT mappings (mapping*)>
```

```
<!ELEMENT mapping ((atom | composite), attributeValue)>
<!ATTLIST mapping
type (node | edge) #REQUIRED>
<!ELEMENT atom EMPTY>
<!ATTLIST atom
property CDATA #REQUIRED
value CDATA #REQUIRED>
<!ELEMENT composite ((atom | composite), connective, (atom | composite))>
<!ELEMENT connective EMPTY>
<!ATTLIST connective
type (and | or) #REQUIRED>
<!ELEMENT attributeValue EMPTY>
<!ATTLIST attributeValue
attribute CDATA #REQUIRED
value CDATA #REQUIRED>
```

This DTD must be included at the beginning of the XML file if the file contains mapping information.

3.1.1.3 Coordinates

The coordinates of node and edge figures calculated by some graph layout algorithm can also be saved in XML format. Currently, the center coordinate of a node figure and bezier control point coordinates of an edge figure are specified in XML. The following XML shows the coordinates of the node and edge figures of the graph defined in XML above:

```
<coordinates>
        <nodeFigure node="u-1" coord="49.0,22.0"/>
        <nodeFigure node="v" coord="144.0,260.0"/>
        <nodeFigure node="w" coord="80.0,102.0"/>
        <nodeFigure node="z" coord="108.0,182.0"/>
        <nodeFigure node="a" coord="148.0,29.0"/>
        <nodeFigure node="b" coord="154.0,182.0"/>
        <edgeFigure edge="e1" coords="46.59765625000001,37.375 39.0,86.0 36.0,182.0
120.91247425920142,243.32567585386772"/>
        <edgeFigure edge="e3" coords="55.15,37.375 61.0,52.0 65.0,64.0
70.24655092527266,77.29126234402412"/>
        <edgeFigure edge="e7" coords="67.66964285714286,37.375 83.0,50.0 108.0,71.0
126.82235221674875,51.23653017241381"/>
        <edgeFigure edge="e2" coords="130.640625,251.23739919354838 51.0,199.0 54.0,103.0
50.688407577819234,49.35220276067163"/>
        <edgeFigure edge="e4" coords="76.74875,115.546875 68.0,152.0 56.0,204.0
120.5166871472147,245.05607363913657"/>
        <edgeFigure edge="e5" coords="81.693359375,115.546875 84.0,134.0 89.0,148.0
94.67187900213833,158.14967821435283"/>
        <edgeFigure edge="e6" coords="105.97348484848486,168.625 103.0,149.0 99.0,135.0
93.78728924749427,125.94634448249005"/>
        <edgeFigure edge="e8" coords="148.0,42.546875 148.0,132.0 149.0,142.0
150.57115349742952,154.56922797943605"/>
        <edgeFigure edge="e9" coords="141.21599264705884,197.5234375 140.0,199.0 137.0,202.0
134.0,204.0 115.0,218.0 83.0,237.0 119.41225756327451,250.72921186812"/>
    </coordinates>
```

The `<nodeFigure>` tag provides the coordinate of the center of the node figure for the node given by the `node` attribute. Similarly, the `<edgeFigure>` tag provides the bezier control point coordinates of the edge figure for the edge given by the `edge` attribute. Coordinates are specified as "$x_1, y_1 \quad x_2, y_2 \quad \ldots \quad x_n, y_n$".

The DTD for the topology XML is given by:

```
<!ELEMENT coordinates (nodeFigure*, edgeFigure*)>
<!ELEMENT nodeFigure (EMPTY) >
```

```
<!ATTLIST nodeFigure
node CDATA #REQUIRED
coord CDATA #REQUIRED>
<!ELEMENT edgeFigure (EMPTY) >
<!ATTLIST edgeFigure
edge CDATA #REQUIRED
coords CDATA #REQUIRED>
```

This DTD <u>must</u> be included at the beginning of the XML file if the file contains coordinate information.

3.1.2 Example of a Complete XML File

The following shows a complete graph XML file containing topology, mapping, and coordinate information:

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE graphFile [
<!ELEMENT graphFile (graph*, mappings*, coordinates*)>
<!ELEMENT coordinates (nodeFigure*, edgeFigure*)>
<!ELEMENT nodeFigure (EMPTY) >
<!ATTLIST nodeFigure
node CDATA #REQUIRED
coord CDATA #REQUIRED>
<!ELEMENT edgeFigure (EMPTY) >
<!ATTLIST edgeFigure
edge CDATA #REQUIRED
coords CDATA #REQUIRED>
<!ELEMENT mappings (mapping*)>
<!ELEMENT mapping ((atom | composite), attributeValue)>
<!ATTLIST mapping
type (node | edge) #REQUIRED>
<!ELEMENT atom EMPTY>
<!ATTLIST atom
property CDATA #REQUIRED
value CDATA #REQUIRED>
<!ELEMENT composite ((atom | composite), connective, (atom | composite))>
<!ELEMENT connective EMPTY>
<!ATTLIST connective
type (and | or) #REQUIRED>
<!ELEMENT attributeValue EMPTY>
<!ATTLIST attributeValue
attribute CDATA #REQUIRED
value CDATA #REQUIRED>
<!ELEMENT graph (nodeProperties*, node*, compositeNode*, edgeProperties*, edge*)>
<!ELEMENT nodeProperties (#PCDATA)>
<!ELEMENT node (#PCDATA)>
<!ATTLIST node
id ID #REQUIRED>
<!ELEMENT edgeProperties (#PCDATA)>
<!ELEMENT edge (#PCDATA)>
<!ATTLIST edge
id ID #REQUIRED
head IDREF #REQUIRED
tail IDREF #REQUIRED
directed (true|false) "false">
]>
<graphFile>
    <coordinates>
        <nodeFigure node="u-1" coord="49.0,22.0"/>
        <nodeFigure node="v" coord="42.0,262.0"/>
        <nodeFigure node="w" coord="80.0,102.0"/>
        <nodeFigure node="z" coord="108.0,182.0"/>
        <nodeFigure node="a" coord="145.0,102.0"/>
        <nodeFigure node="b" coord="154.0,182.0"/>
        <edgeFigure edge="e1" coords="46.30384824371398,39.255371240230595 39.0,86.0 36.0,182.0
40.983984375000006,248.45312500000006"/>
```

```
        <edgeFigure edge="e3" coords="55.653276666647095,38.63319166661774 61.0,52.0 65.0,64.0
74.65254934210526,88.453125"/>
        <edgeFigure edge="e7" coords="64.50080770148686,34.84352638123198 84.0,51.0 109.0,72.0
121.59384464483149,82.49487053735959"/>
        <edgeFigure edge="e2" coords="43.93526785714286,248.453125 51.0,199.0 54.0,103.0
50.07136821892032,39.35616514650896"/>
        <edgeFigure edge="e4" coords="76.74875,115.546875 68.0,152.0 56.0,204.0
45.26993534482756,248.453125"/>
        <edgeFigure edge="e5" coords="81.693359375,115.546875 84.0,134.0 89.0,148.0
94.67187900213835,158.14967821435278"/>
        <edgeFigure edge="e6" coords="105.97348484848486,168.625 103.0,149.0 99.0,135.0
93.78728924749427,125.94634448249002"/>
        <edgeFigure edge="e8" coords="146.3546875,115.546875 148.0,132.0 149.0,142.0
150.5711534974295,154.569227979436"/>
        <edgeFigure edge="e9" coords="141.21599264705884,197.5234375 140.0,199.0 137.0,202.0
134.0,204.0 115.0,218.0 83.0,237.0 65.60492910952841,247.60675054297047"/>
    </coordinates>
    <mappings>
        <mapping type="node">
            <atom property="type" value="dna"/>
            <attributeValue attribute="node shape" value="ellipse"/>
        </mapping>
        <mapping type="edge">
            <atom property="type" value="regulate"/>
            <attributeValue attribute="connector end" value="filled rectangle head"/>
        </mapping>
    </mappings>
    <graph>
        <nodeProperties>"type" "location" </nodeProperties>
        <node id="u-1">"dna" "loc1" </node>
        <node id="v">"rna" "loc1" </node>
        <node id="w">"molecule" "loc2" </node>
        <node id="z">"molecule" "loc1" </node>
        <node id="a">"enzyme" "loc2" </node>
        <node id="b">"enzyme" "loc1" </node>
        <edgeProperties>"type" "strength" </edgeProperties>
        <edge id="e1" tail="u-1" head="v" directed="true">"regulate" "1.0" </edge>
        <edge id="e3" tail="u-1" head="w" directed="true">"regulate" "1.0" </edge>
        <edge id="e7" tail="u-1" head="a" directed="true">"convert" "-1.0" </edge>
        <edge id="e2" tail="v" head="u-1" directed="true">"regulate" "1.0" </edge>
        <edge id="e4" tail="w" head="v" directed="true">"regulate" "-1.0" </edge>
        <edge id="e5" tail="w" head="z" directed="true">"catalyze" "1.0" </edge>
        <edge id="e6" tail="z" head="w" directed="true">"convert" "-1.0" </edge>
        <edge id="e8" tail="a" head="b" directed="true">"convert" "-1.0" </edge>
        <edge id="e9" tail="b" head="v" directed="true">"convert" "-1.0" </edge>
    </graph>
</graphFile>
```

3.1.3 Opening a Graph XML File

To open a graph XML file in FCModeler, either select the “open graph file” menu item from the file menu or click the toolbar button. The open graph dialog box will then be shown as in Figure 1.

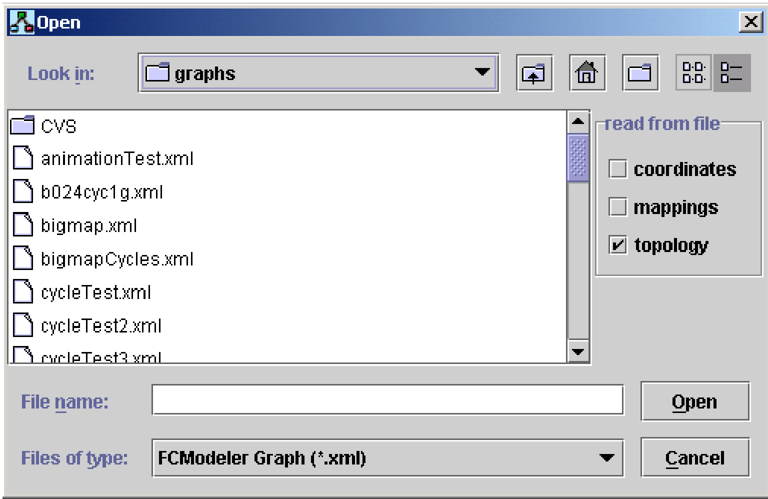

Figure 1. The open graph dialog box.

Since any graph XML can contain topology, mappings, or coordinates in any combination, the dialog box allows the user to select which information is read from a specific file. The checkboxes on the right side of the dialog configure the information to read from the XML file. For instance, if a graph is already open in FCModeler, selecting only the mappings from an XML file that contains both mappings and topology will only read the mappings and apply them to the current graph. Also, reading a file of coordinates for an already open graph will reposition the node and edge figures appropriately.

3.1.4 Saving a Graph XML File

To save the current graph to an XML file, either select the "save graph" menu item from the file menu or click the toolbar button. The save graph dialog box will then be shown as in Figure 2.

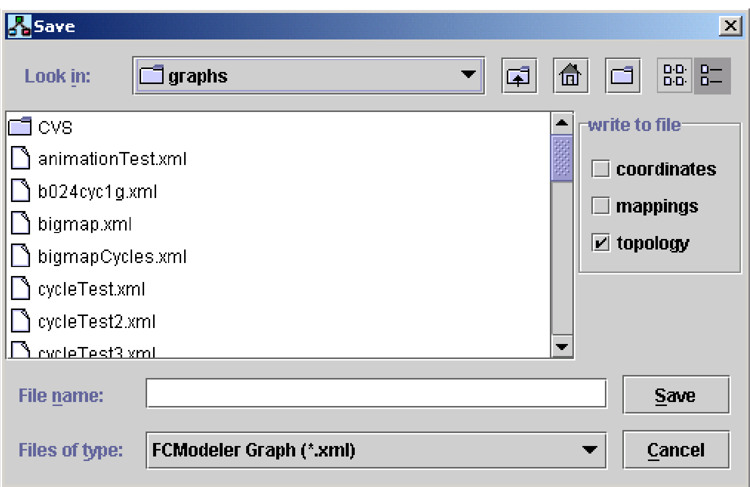

Figure 2. The save graph dialog box.

FCModeler can write the current topology, mappings, or coordinates in any combination to a graph XML file. The checkboxes on the right side of the dialog control the information that is written to the file.

3.1.5 Saving a JPEG Image of the Graph

FCModeler can also save the current graph view in JPEG format. This is useful for creating an image file of the entire graph, which can then be viewed in a web browser or inserted into some other document. To create a JPEG image of the graph view, select the “save as jpg” menu item from the file menu.

3.2 MySQL Database

To be written…

4 Graph Layout

To view a graph, a graph layout algorithm must compute positions of the node and edge figures. Many different graph layout algorithms exist. FCModeler currently uses Dot and GEM to compute its layout.

4.1 Simple Dot Layout

There are two types of layout that can be computed by Dot. The first is a generic layout called the Simple Dot Layout. When a graph is opened in FCModeler, either select the “dot layout” menu item from the layout menu or click the toolbar button to use the simple dot layout. Figure 3 below shows a graph layout done using the simple dot layout.

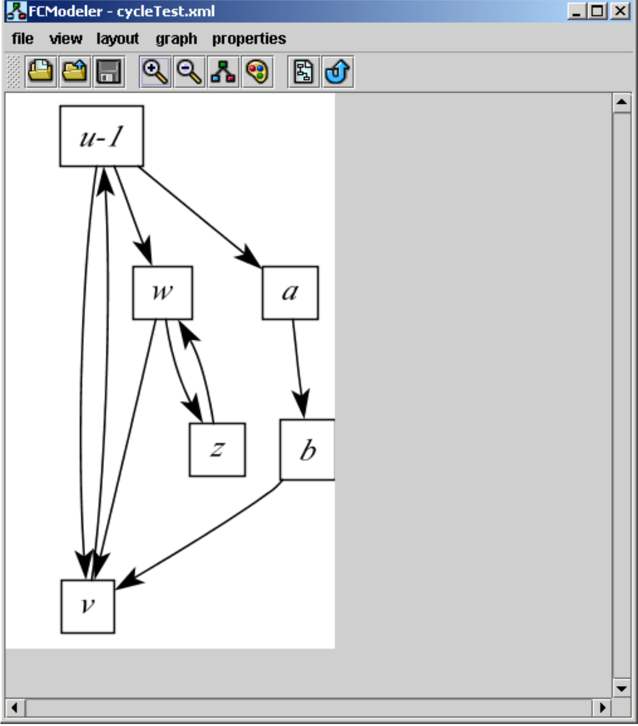

Figure 3. Simple dot layout.

4.2 Rank-Cluster Dot Layout

Dot can also be used to compute a more customized layout. Based on their values of certain node properties, the node figures can be placed on horizontal ranks or into clusters. When a graph is open in FCModeler, select the "dot rank-cluster layout" menu item from the layout menu. An open file dialog box is then shown, and a layout XML file must be selected (see below). Figure 4 below shows the same graph as in Figure 3 using the rank-cluster layout.

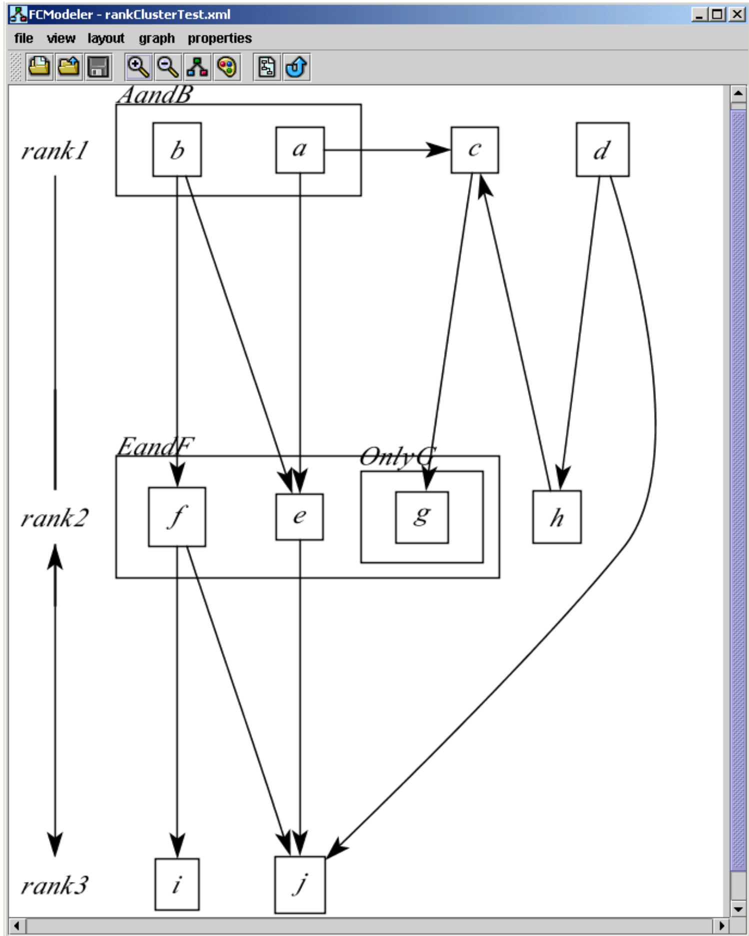

Figure 4. Rank-cluster dot layout.

4.2.1 Layout XML File

The ranks and clusters for the rank-cluster layout are specified in a layout XML file. The XML file used in the layout for Figure 4 is shown below:

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE layout [
<!ELEMENT layout (rank*, cluster*)>
<!ELEMENT rank (atom*, composite*)>
<!ATTLIST rank
```

```
label CDATA #REQUIRED>
<!ELEMENT cluster (atom*, composite*, cluster*)>
<!ATTLIST cluster
label CDATA #REQUIRED>
<!ELEMENT atom EMPTY>
<!ATTLIST atom
property CDATA #REQUIRED
value CDATA #REQUIRED>
<!ELEMENT composite ((atom | composite), connective, (atom | composite))>
<!ELEMENT connective EMPTY>
<!ATTLIST connective
type (and | or) #REQUIRED>
]>

<layout>

<rank label="rank1">
    <atom property="type" value="type1"/>
</rank>

<rank label="rank2">
    <atom property="type" value="type2"/>
</rank>

<rank label="rank3">
    <atom property="type" value="type3"/>
</rank>

<cluster label="AandB">
    <composite>
        <atom property="label" value="a"/>
        <connective type="or"/>
        <atom property="label" value="b"/>
    </composite>
</cluster>

<cluster label="EandF">
    <composite>
        <atom property="label" value="e"/>
        <connective type="or"/>
        <atom property="label" value="f"/>
    </composite>
    <cluster label="OnlyG">
        <atom property="label" value="g"/>
    </cluster>
</cluster>

</layout>
```

The `<rank>` tag is used to specify the nodes that should be placed in a certain horizontal rank. Each rank has a label, specified by the `label` attribute, which is shown on the left-hand side of the graph view. Nodes are selected using the same type of XML as in the property-to-visual-attribute mappings (see the Mappings section above).

Similarly, the `<cluster>` tag is used to specify nodes to place in a cluster. Each cluster is surrounded by a rectangle and has a label, specified by the `label` attribute, which is shown on the upper-left of the cluster. The `<cluster>` tag uses the same node selection mechanism as the `<rank>` tag. `<cluster>` tags can be nested inside each other, creating nested clusters as shown in clusters EandF and OnlyG above.

4.3 GEM Layout(s)

Additional layout options are also featured GEM, AGEM,FastGEM. These algorithms are modeled off code developed by the Tulip project (www.tulip.org).

GEM: This is the standard layout as detailed in "A Fast Adaptive Layout Algorithm for Undirected Graphs" by Frick, Ludwig and Mehldau. This code is a Java translation of the Frick's implementation in C code. GEM turns every node into an electron and every edge into a stretched spring and places a gravitational force at the barycenter of the layout. A node is attracted to all nodes adjacent to it and is repelled by every node whether or not it is connected by an edge. In addition, there is an attractive gravitational force between each node and the barycenter of the graph. GEM has three main loops: the insertion loop, the arrangement loop, and the optimization loop. The insertion loop is where the nodes are initially placed. The closer a placement is to the final layout, the faster the arrangement loop runs. The arrangement loop does most of the work getting the nodes to their final positions. Following the example of Tulip, we did not implement the optimization loop.

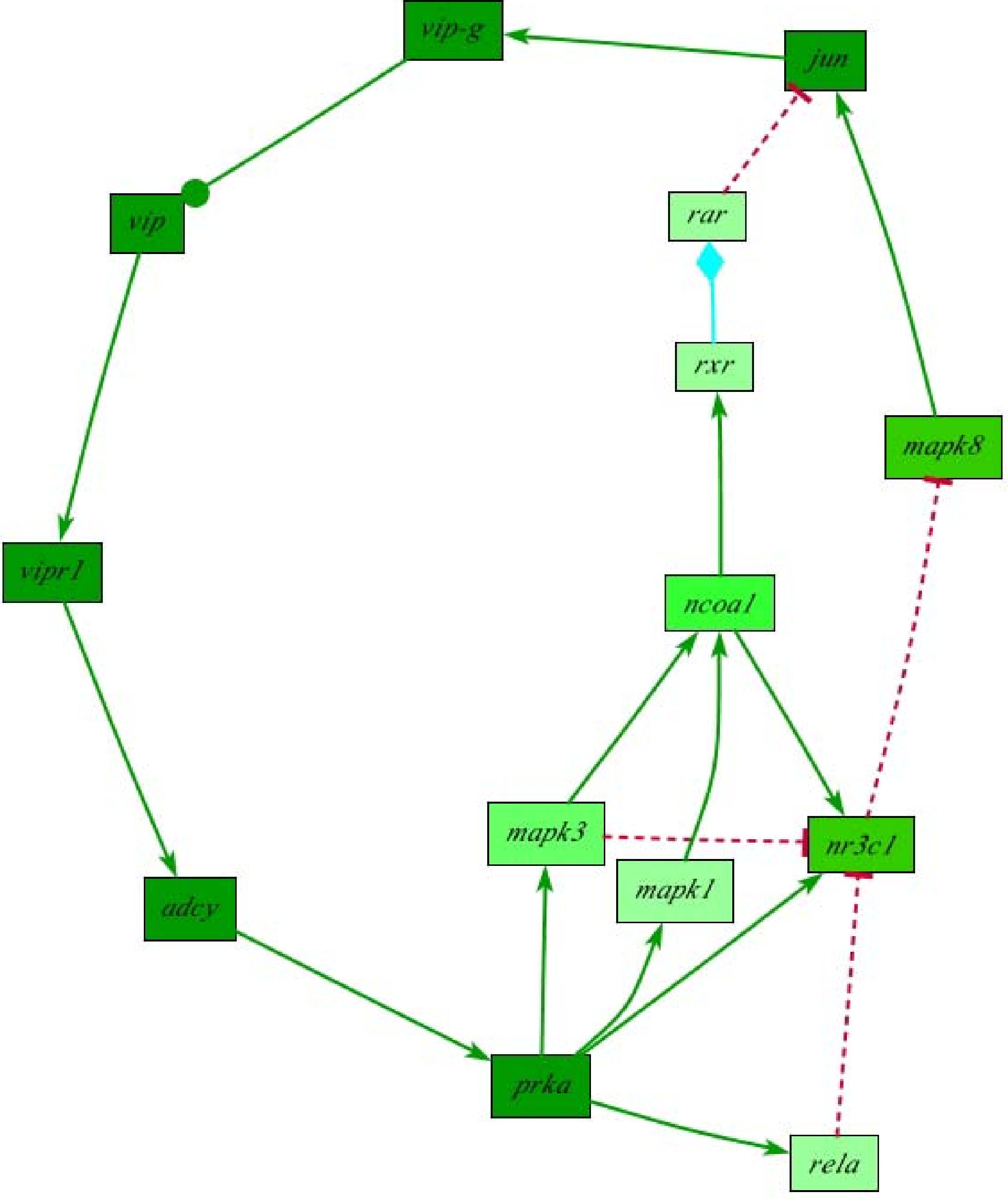

Figure 5: Example of a GEM layout.

AGEM: This is the "new" spring embedder algorithm that differs significantly from GEM. It uses much of the framework of the GEM algorithm, but the heart of it is different. AGEM removes the repulsive force between pairs of nodes, reverses the direction of the gravitational force to make it repulsive, and retains the attractive spring forces between adjacent nodes. AGEM gives a layout that is similar to the regular GEM, but does it much faster. Because of the lack of repulsion between nodes, an unlucky initial random layout could leave many nodes bunched together. Also, the reversed gravity tends to push singletons and disconnected subgraphs away from the barycenter. This increases the global temperature of the layout causing it to use its maximum allowed number of iterations instead of stopping early due to settling of the graph.

FastGEM: FastGEM is a compromise designed to give the more consistent quality of GEM while retaining some of the speed of AGEM. FastGEM uses AGEM to do an initial layout and GEM to finish. AGEM replaces Frick's insertion loop. With a layout that is often near to the final layout, the regular GEM algorithm runs significantly faster. If there are disconnected pieces of the graph, the AGEM algorithm will take a long time to complete, so any speed advantages over GEM will be lost.

In summary:
GEM: The standard GEM layout. This one can handle disconnected graphs and takes the longest to run, though the quality is good and consistent.
AGEM: Runs much faster than GEM. A disconnected graph will cause it to blow up and take a long time to complete. This is visually close to GEM, but the quality is fair at best and much more inconsistent.
FastGEM: This uses AGEM to do the initial layout, then GEM can polish it up. AGEM gets the initial placement close, making GEM much faster, but GEM makes the quality much more consistent.

5 Interacting with the Graph View

FCModeler supports a great deal of interaction with the graph view. This interaction is done using the mouse and keyboard keys.

5.1 Selecting Node and Edge Figures

One basic form of interaction with the graph view is selection. Multiple nodes and/or edges can be selected at any given time. Node and edge figures can be selected in several ways, as shown in Table 1.

Table 1. Methods of selecting nodes and edges.

Selection	**5.1.1.1.1 User Action**
Individual node or edge	Left-click on a node or edge figure
Multiple nodes and/or edges	Ctrl + left-click on multiple node and/or edge figures
Multiple nodes and/or edges	Left-click & drag around multiple node and/or edge figures

Selecting a node or edge highlights the figure in the graph view. Highlighted nodes have a yellow border placed around them, while a thick red line denotes the highlighted edges.

Selecting not only visually highlights the node and edges figures, but also creates a subset of the nodes and edges in the graph. Thus, selection is the starting point for other forms of interaction.

5.2 Node Figures

FCModeler supports several different forms of interaction with nodes, including adding new nodes, deleting nodes, and moving nodes. These interactions are described in more detail below.

Table 2. Node interactions.

Behavior	User Action
Add a new node	Ctrl + left-click on an empty portion of graph
Delete a node	Ctrl + right-click on a node
Move a node(s)	Left-click and drag the node(s)

5.2.1 Adding

To add a new node to the graph, hold down the Ctrl key and left-click with the mouse in an empty portion of the graph. A dialog box will appear, asking for a unique node name for the new node. Enter the node name and click ok. The new node will then appear in the graph view.

5.2.2 Deleting

To delete an existing node from the graph, hold down the Ctrl key and right-click with the mouse on a node in the graph view. The node will be removed from the graph along with any edges going into or coming out from that node.

5.2.3 Moving

To re-position a node figure in the graph view, left-click and drag the node figure. The node figure, along with all edges incident with that node, will then move around the graph view.

5.3 Edge Figures

FCModeler supports several different forms of interaction with edges, including adding new edges, modifying edges, deleting edges, and moving edges. These interactions are described in more detail below.

Table 3. Edge interactions.

Behavior	User Action
Add a new edge	Ctrl + left-click on tail node, release mouse on empty portion of graph or on head node
Modify the tail or head node of an edge	Left-click to select the edge, then left-click and drag the blue rectangle on either end of the edge, snap blue rectangle to new tail or head node
Delete an edge	Ctrl + right-click on an edge
Moving	To be written…

5.3.1 Adding

To add a new edge to the graph, hold down Ctrl and left-click on the tail node of the edge. Then, release the mouse button on either an empty portion of the graph or on the head node of the edge. If the button is released on an empty portion of the graph, the head end of the edge can be placed on the head node as described in the next section.

5.3.2 Modifying

To move the tail or head end of an edge to a new tail or head node, first select the edge by left-clicking on it. Then, left-click and drag the blue rectangle at the desired end of the edge until it snaps to the new tail or head node.

5.3.3 Deleting

To delete an existing edge, hold down Ctrl and right-click on the edge. The edge will then be removed from the graph.

5.3.4 Moving

Nodes and edges can be moved in the graph by first selecting the element. In the figure below, the highlighted node has been selected and moved. The edges move along with the node, unfortunately the result is often very strange looking. The edges can be moved by selecting the edge then manipulating the control points which are shown as blue boxes. Once an acceptable view has been created, the coordinates of the nodes and edges can be saved.

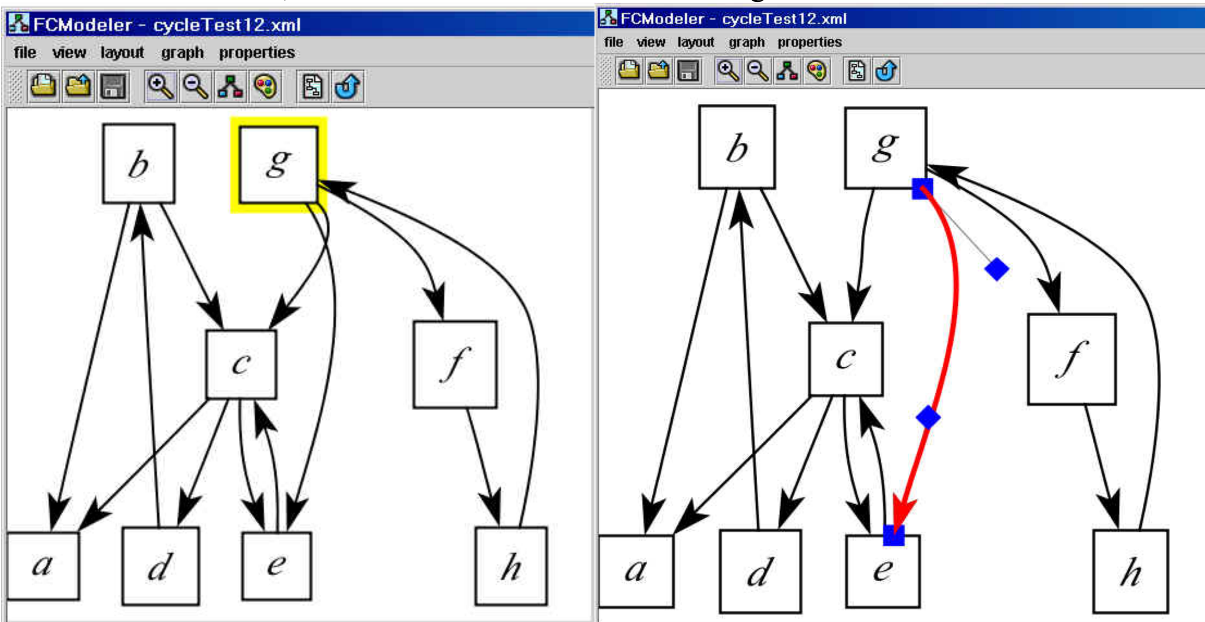

Figure 6 shows the manipulation of node and edge objects in FCModeler.

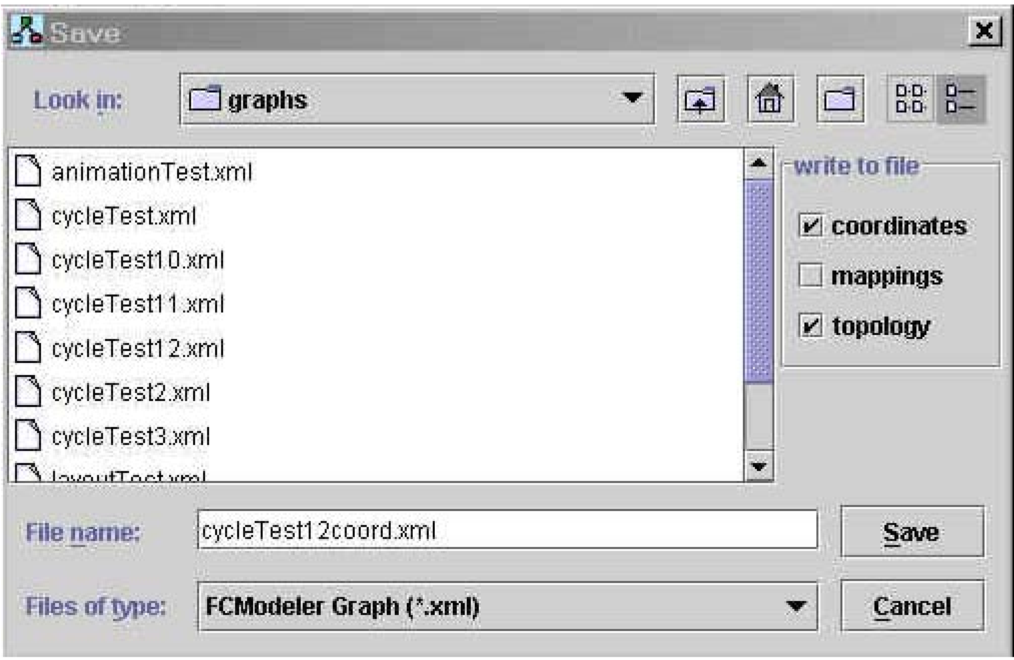

Figure 7 shows how to save graph coordinates using the save function under the File menu.

5.4 Zooming

FCModeler supports zooming of the graph view, to examine the graph in varying levels of detail. To zoom in, either select the "zoom in" menu item from the view menu or click the toolbar button. To zoom out, either select the "zoom out" menu item from the view menu or click the toolbar button.

5.5 Panning

For large graphs, the FCModeler window may not be large enough to view the entire graph. Thus, FCModeler supports panning to view different areas of a large graph. The scrollbars on the bottom and left side of the FCModeler window control panning in the horizontal and vertical directions, respectively.

5.6 Finding a Particular Node

To find a particular node in a graph, select the "find node" menu item from the view menu. A dialog box will appear asking for the label of the node. After entering it and clicking ok, the view will center on the node and it will be selected.

6 Mappings

6.1 Node and Edge Properties

The nodes and edges of the graph can have multiple properties associated with them. For example, node properties could be "type" or "organelle" while edge properties could be "type" or "strength". In this way, the properties serve to provide information about what the nodes and edges of the graph represent.

Each individual node and edge of the graph then has a specific value for each property. For example, values for the node property "type" could be "gene", "RNA", "protein", "environmental factor", etc. Values for the edge property "type" could be "conversion", "regulation", "catalyst", etc. FCModeler does not restrict the properties or values that a graph may possess. They are determined from either the graph XML file or the MySQL database (see the Graph XML Files and MySQL Database sections for more details).

6.1.1 Property Viewer

The Property Viewer can be used to observe the property values of specific nodes and edges in FCModeler. To open the Property Viewer, select the "property viewer" menu item from the properties menu. The Property Viewer, as seen in Figure , will then be shown.

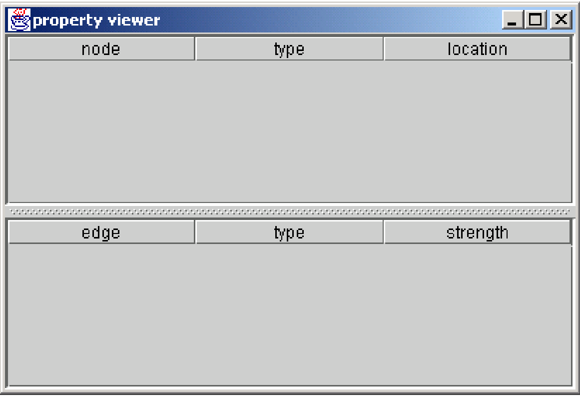

Figure 8. Empty property viewer.

The Property Viewer shows the property values of all selected nodes and edges. Figure 8 shows an example of how it works. When the Property Viewer is open, simply select any nodes and edges as described in the Selecting Node and Edge Figures section above. The property values for those nodes and edges are then shown in the Property Viewer. The Property Viewer can be closed at any time.

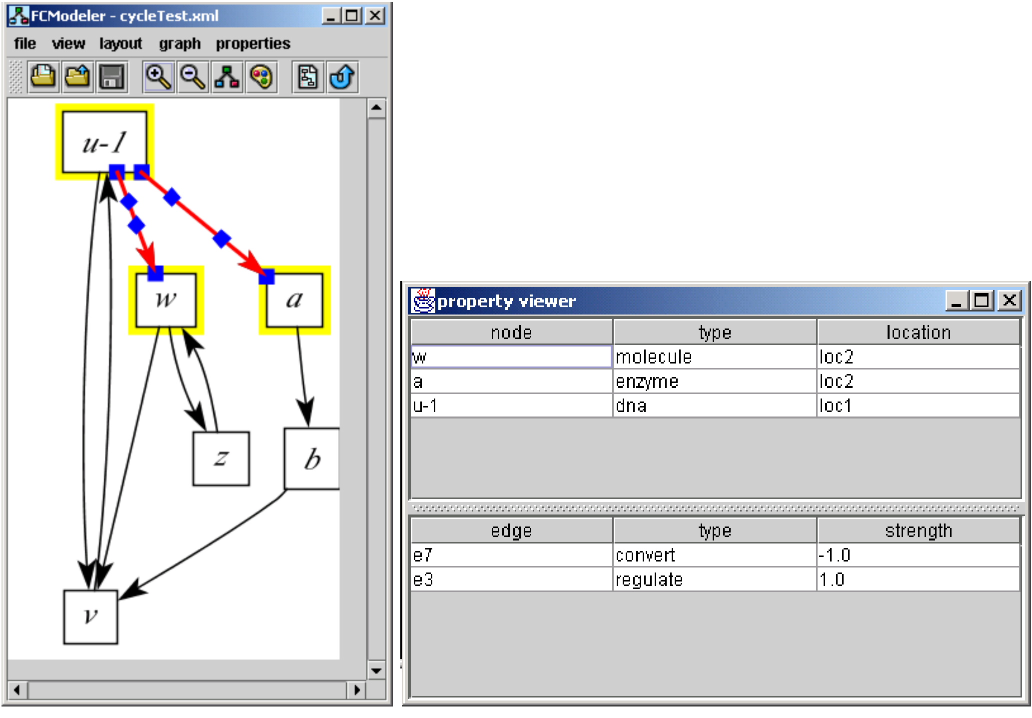

Figure 8. Property viewer showing property values of selected nodes and edges.

6.2 Visual Attributes of Node and Edge Figures

The node and edge figures in FCModeler have different visual attributes associated with them. For instance, node figures can have different shapes and edges can have different colors. Table 4 lists the different visual attributes of the node and edge figures.

Table 4. The visual attributes of node and edge figures supported by FCModeler.

Visual Attributes of Node Figures	Visual Attributes of Edge Figures
Shape	Line color
Fill color	Line width
Outline color	Dash pattern
	Arrowhead shape

Each visual attribute then has specific values. Node shapes can be ellipses, rectangles, round rectangles, diamonds, etc. Colors (for both nodes and edges) can be specified by any RGB value.

6.3 Using Visual Attributes to Visualize Properties

FCModeler allows the user to assign property values to visual attribute values. For example, the value “gene” of the node property “type” can be assigned to the value “ellipse” of the visual attribute “node shape”. Then, all nodes in the graph that are of type gene will be shown using an elliptical node shape. Any property value can be assigned to any visual attribute value. This mapping ability allows the user to visualize the property values of the nodes and edges in any

way they want. These mappings can either be specified in a graph XML file (see the Mappings section above) or created by the user using the Mapping Editor.

6.4 Creating Mappings Using the Mapping Editor

The Mapping Editor is used to assign property values to visual attribute values, so the property values can be seen in the graph view. To open the Mapping Editor, either select the "mapping editor" menu item from the view menu or click the toolbar button.

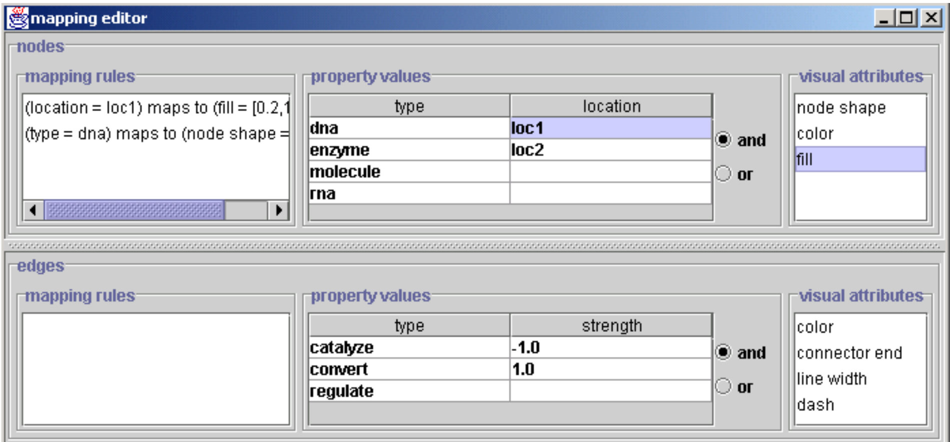

Figure 9. Mapping editor.

Figure 9 shows the Mapping Editor. The property values of the nodes and edges are shown in the middle of the window in a table; the columns represent the different properties, and the entries in the table represent the different values that are present in the graph. Note that there is not an entry for each individual node or edge in the table, only for the different property values that the nodes and edges possess.

The visual attributes are shown in a list on the right side of the window. To create a mapping, first select a property value from the table. Then, double click on the desired visual attribute. Another window will open, allowing you to select the value of that visual attribute. Clicking ok creates the mapping, applies it to the graph view, and adds it to the list of mappings on the left side of the Mapping Editor. To delete a mapping rule, simply left-click on the rule in the list and press the "Delete" key on the keyboard.

Mapping rules can be created with multiple property values. For instance, nodes with a value of "molecule" for the property "type" and a value of "loc1" for the property "location" can be assigned to the value of "ellipse" for the visual attribute "node shape". Thus, only nodes with both property values will be shown as ellipses. Nodes with only one property value but not the other will be unaffected by the mapping rule. To create this type of mapping, select multiple property values from the table by holding down Ctrl and left-clicking. The "and" and "or" radio buttons determine how the property values are combined (in the example given in this paragraph the "and" button would be selected). Then, configure the visual attribute the same way as before.

7 Animation

FCModeler has animation functionality, allowing the visual attributes of the node and edge figures to change over time. More specifically, property value to visual attribute value mapping rules can be defined for discrete time steps. FCModeler then steps through the sets of mappings and the user can see things changing in the graph view. Animation can be used to show the progression of some modeling algorithm, such as fuzzy cognitive maps, among other things.

7.1 XML File Format

The mapping rules for the animation are specified in an XML file. The following code segment shows an example of a complete animation XML file:

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE animation [
<!ELEMENT animation (time*)>

<!ELEMENT time (mapping*)>
<!ATTLIST time
n ID #REQUIRED>

<!ELEMENT mapping ((atom | composite), value)>
<!ATTLIST mapping
type (node | edge) #REQUIRED>

<!ELEMENT atom EMPTY>
<!ATTLIST atom
property CDATA #REQUIRED
value CDATA #REQUIRED>

<!ELEMENT composite ((atom | composite), connective, (atom | composite))>

<!ELEMENT connective EMPTY>
<!ATTLIST connective
type (and | or) #REQUIRED>

<!ELEMENT value EMPTY>
<!ATTLIST value
attribute CDATA #REQUIRED
value CDATA #REQUIRED>
]>

<animation>

<time n="n0">
       <mapping type="node">
            <atom property="label" value="a"/>
            <attributeValue attribute="fill" value="[1.0,0.0,1.0]"/>
        </mapping>
       <mapping type="edge">
            <atom property="label" value="e8"/>
            <attributeValue attribute="color" value="[1.0,0.0,0.0]"/>
        </mapping>
</time>
<time n="n1">
       <mapping type="node">
            <atom property="label" value="b"/>
            <attributeValue attribute="fill" value="[1.0,0.0,1.0]"/>
        </mapping>
       <mapping type="edge">
            <atom property="label" value="e9"/>
            <attributeValue attribute="color" value="[1.0,0.0,0.0]"/>
        </mapping>
</time>
<time n="n2">
       <mapping type="node">
```

```
            <atom property="label" value="v"/>
            <attributeValue attribute="fill" value="[1.0,0.0,1.0]"/>
        </mapping>
        <mapping type="edge">
            <atom property="label" value="e2"/>
            <attributeValue attribute="color" value="[1.0,0.0,0.0]"/>
        </mapping>
</time>
<time n="n3">
        <mapping type="node">
            <composite>
                <atom property="label" value="u-1"/>
                <connective type="or"/>
                <atom property="label" value="b"/>
            </composite>
            <attributeValue attribute="fill" value="[1.0,0.0,1.0]"/>
        </mapping>
        <mapping type="edge">
            <atom property="label" value="e3"/>
            <attributeValue attribute="color" value="[1.0,0.0,0.0]"/>
        </mapping>
</time>
<time n="n4">
        <mapping type="node">
            <atom property="label" value="w"/>
            <attributeValue attribute="fill" value="[1.0,0.0,1.0]"/>
        </mapping>
        <mapping type="edge">
            <atom property="label" value="e5"/>
            <attributeValue attribute="color" value="[1.0,0.0,0.0]"/>
        </mapping>
</time>
<time n="n5">
        <mapping type="node">
            <atom property="label" value="z"/>
            <attributeValue attribute="fill" value="[1.0,0.0,1.0]"/>
        </mapping>
        <mapping type="edge">
            <atom property="label" value="e6"/>
            <attributeValue attribute="color" value="[1.0,0.0,0.0]"/>
        </mapping>
</time>
<time n="n6">
        <mapping type="node">
            <atom property="label" value="w"/>
            <attributeValue attribute="fill" value="[1.0,0.0,1.0]"/>
        </mapping>
        <mapping type="edge">
            <atom property="label" value="e4"/>
            <attributeValue attribute="color" value="[1.0,0.0,0.0]"/>
        </mapping>
</time>
<time n="n7">
        <mapping type="node">
            <atom property="label" value="v"/>
            <attributeValue attribute="fill" value="[1.0,0.0,1.0]"/>
        </mapping>
</time>

</animation>
```

The mapping rules are specified at individual time points. A `<time>` tag represents each time point and the `n` attribute defines which point in time the tag represents. The `<mapping>` tags within the `<time>` tags define the mapping rules in exactly the same way as in the graph XML files (see the Mappings section above).

To open an animation XML file, select the “open animation file” menu item from the file menu.

7.2 Controls

After FCModeler reads the XML file the animation control window in Figure 10 is shown. The controls work just like a cassette tape or CD player. The user can play the animation, pause it, step forwards or backwards, and adjust the time delay between steps.

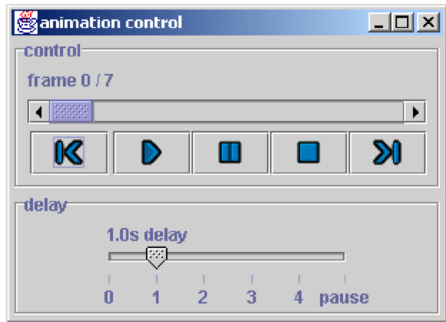

Figure 10. Animation control.

8 Graph Theoretic Operations

Graphs are well-studied mathematical objects and an entire area of mathematics, called graph theory, exists to study their properties. A graph consists of two parts: a set of objects called nodes (or vertices) and a set of relations between nodes called edges. If a graph is undirected, its edges are unordered pairs of vertices, $e = \{u, v\} = \{v, u\}$, while a directed graph (called a digraph) has ordered pairs for edges, $e = (u, v) \neq (v, u)$. The first node of a directed edge is called the tail, and the second node is called the head. A graph is typically denoted $G = (V, E)$, where V is the node set (or vertex set) and E is the edge set. FCModeler implements several graph theoretic algorithms that are useful for analyzing the properties of a specific graph. These algorithms, described below, are also closely coupled with the graph view to visually present their results.

8.1.1 Subgraph Creation

For a digraph $D = (V, E)$, a digraph H is a subdigraph of D if

- $V(H) \subseteq V(D)$
- $E(H) \subseteq E(D)$
- $E(H) = \{(v, u) \in E(D) | v, u \in V(H)\}$

In other words, all of the nodes of H must be in D, all of the edges of H must be in D, and the edges of H must have both end-nodes in H.

For a given digraph $D = (V, E)$, a subdigraph H can be created in several ways:

- Given a set of nodes $W \subseteq V(D)$
    - $V(H) = W$
    - $E(H) = \{(v, u) \in E(D) | v, u \in V(H)\}$
    - H consists of some subset of nodes of D and all edges in D with both end-nodes in that set
- Given a set of nodes $W \subseteq V(D)$ and an integer p

- $V(H) = N_D^p[W]$
- $E(H) = \{(v,u) \in E(D) | v,u \in V(H)\}$
- $N_D^p[W]$ is defined as the closed *p*-th neighborhood of *W*. Saving *W* and all nodes visited on directed paths of length *p* from each node in *W* forms the set.
- $N_D^p[W] = N_D^{+p}[W] \cup N_D^{-p}[W] = \bigcup_{i=0}^{p} N_D^{+i}(W) \cup \bigcup_{i=0}^{p} N_D^{-i}(W)$
- $N_D^{+p}(W) = N_D^{+}\left(N_D^{+(p-1)}(W) - \bigcup_{i=0}^{p-1} N_D^{+i}(W) \right)$ and $N_D^{-p}(W)$ is defined similarly.

* Given a set of edges $B \subseteq E(D)$
  - $V(H) = \{v \in V(D) | v \in (v,u) \in E(H)\}$
  - $E(H) = B$
  - *H* consists of all end-nodes of the edges in *B* and all of the edges in *B*.
* Given a set of nodes $W \subseteq V(D)$ and a set of edges $B \subseteq E(D)$
  - $V(H) = W$
  - $E(H) = \{(v,u) \in B | v,u \in V(H)\}$
  - *H* consists of all nodes in *W* and all edges in *B* with both end-nodes in *W*.

In FCModeler, the selected node and edge figures in the graph view represent the node and edge sets *W* and *B* (see the Selecting Node and Edge Figures section above). To create a subgraph when some nodes and/or edges are selected, either select the "create subgraph" menu item from the graph menu or click the toolbar button. The subgraph creation dialog box is then shown as in Figure 11 below. This dialog allows the user to select one of the subgraph creation algorithms described above. Making a selection and clicking the ok button opens a new FCModeler window showing the subgraph.

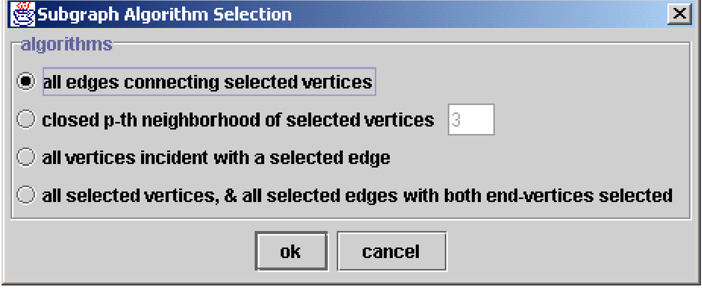

Figure 11. The subgraph creation dialog box.

8.1.2 Strongly Connected Components

A digraph *D* is strongly connected if every vertex of *D* is reachable from every other vertex of *D*. The strongly connected components (SCC's) of a digraph *D* are the maximal strongly connected subdigraphs of *D*. The SCC's are useful to analyze, because every node in an SCC is reachable from every other node in the same SCC.

The algorithm for finding the SCC's of a digraph *D* is extremely simple and is based on two modifications of depth-first search (DFS).

1. Compute an acyclic ordering of the nodes using DFS.
2. Compute the converse *D'* of *D*.
3. Perform DFS on *D'*, using the ordering from Step 1. Each DFS tree is a SCC of *D*.

To find the strongly connected components of a graph in FCModeler, select the "find strongly connected components" menu item from the graph menu. After performing the above algorithm, FCModeler shows all of the SCC's to the user in a list. Clicking on an SCC in the list selects all of the node figures in that SCC in the graph view. A subgraph can then be created based on those nodes, since their node figures are selected.

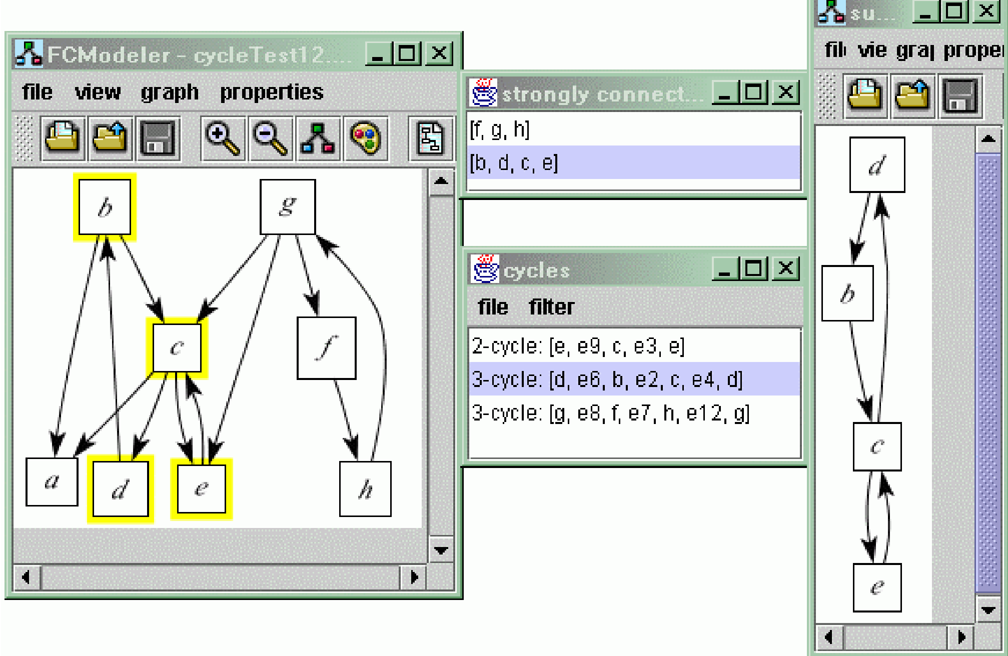

Figure 12 Example of selecting cycles and strongly connected components as subgraphs.

8.1.3 Cycle Search

A path in a digraph is an alternating sequence of nodes and edges, $v_1 e_1 v_2 e_2 v_3 ... v_{k-1} e_{k-1} v_k$, such that tail($e_i$) = v_i, head(e_i) = v_{i+1}, and all nodes are distinct. A path can be viewed as starting at node v_1 and following edges through the graph until node v_k is reached. If $v_1 = v_k$, the path is called a cycle.

Often it is useful to know all of the cycles of a digraph, and finding all of the cycles in a digraph is a well-studied problem in graph theory. FCModeler implements a cycle search algorithm that finds all of the cycles in the graph. To perform the cycle search, either select the "find cycles" menu item from the graph menu or click the toolbar button.

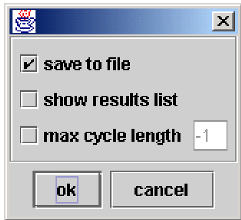

Figure 13. The cycle search dialog box.

Before performing the cycle search algorithm, the user must select what to do with the results. This is done through the dialog shown in Figure 13. The results can be shown in a list, saved to a file, or both. If the list option is chosen, when the algorithm finishes a list of cycles is shown. Selecting a cycle from the list selects the nodes and edges of the cycle.

8.1.4 Cycle XML Files

The cycles can be saved to an XML file for later use. The following XML shows an example of a cycle XML file:

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE cycles [
<!ELEMENT cycles (cycle*)>
<!ELEMENT cycle ((node, edge), (node, edge), (node, edge)*)>
<!ELEMENT node EMPTY>
<!ATTLIST node
id CDATA #REQUIRED>
<!ELEMENT edge EMPTY>
<!ATTLIST edge
id CDATA #REQUIRED>
]>

<cycles>
        <cycle>
                <node id="b"/>
                <edge id="e9"/>
                <node id="v"/>
                <edge id="e2"/>
                <node id="u-1"/>
                <edge id="e7"/>
                <node id="a"/>
                <edge id="e8"/>
        </cycle>
        <cycle>
                <node id="z"/>
                <edge id="e6"/>
                <node id="w"/>
                <edge id="e5"/>
        </cycle>
        <cycle>
                <node id="w"/>
                <edge id="e4"/>
                <node id="v"/>
                <edge id="e2"/>
                <node id="u-1"/>
                <edge id="e3"/>
        </cycle>
        <cycle>
                <node id="v"/>
                <edge id="e2"/>
                <node id="u-1"/>
                <edge id="e1"/>
        </cycle>
</cycles>
```

This XML is extremely simple. Each cycle is surrounded by a `<cycle>` tag, and consists of alternating `<node>` and `<edge>` tags. The nodes and edges are identified by the `id` attributes in the `<node>` and `<edge>` tags.

These cycle XML files can also be re-opened in FCModeler at a later time. To do this, select the "open cycle file" menu item from the graph menu. The cycles in the XML file are then shown in a list. Note that the cycles must be present in the current graph in FCModeler.

8.2 Alternate Paths between Nodes

One common problem is to find alternate paths between nodes. This feature is being added into FCModeler and should be available with the next release.

8.3 Clustering Cycles

Cycles obtained from directed graphs may be similar to each other based on their node and edge content. Thus, clustering algorithms may be used to find natural groups of similar cycles. Prior to clustering, a distance metric and generalized set median must be defined for the objects to be clustered. Several possible representations for the cycles are discussed, including strings, graphs, and weighted sets. Weighted sets are chosen because of the simplicity of the associated metric and median. Self-organizing maps are used to find the clusters of cycles obtained from a directed graph. This approach is implemented in FCModeler, a bioinformatics tool, and uses the JSOMap package for self-organizing maps. An example with real-world data is used to show the effectiveness of the SOM in finding clusters of similar cycles. Once the SOM algorithm is finished, the results are viewed in another window. This map view window shows a grid of small graphs, one for each unit of the map. Each graph represents the model for that particular map unit. The models are just the generalized median of the set of cycles assigned to that map unit, as presented earlier. Figure 14 below shows an example of a 4-by-4 SOM applied to a larger graph. Node color and edge thickness are used to show the weights of the nodes and edges of the models. Brighter red corresponds to higher weights of the nodes, while thicker lines correspond to higher weights of the edges.

As Figure 14 shows, it can be a bit difficult to fully examine the model graphs in the map view. Clicking on one of the graphs in the map view puts it in an additional FCModeler window for easier analysis. A good cluster of cycles can be seen in Figure 15 below, which is a view of the model graph in the bottom-left map unit of Figure 14. This model represents ten individual cycles. All cycles contain the nodes ACC_synthase, ACC, ethylene, CTR1, EIN2, EIN3, senescence, NIT2, and IAA. Note the dark red color of these nodes and the edges between them, showing the maximum weight value. Also, note the five different paths between ethylene and CTR1, and the two different paths between CTR1 and EIN2. There are two cycles for each of the five different paths between ethylene and CTR1: one path going from CTR1 directly to EIN2 and another going through MAPKK and MAPK. This is where the ten cycles come from. Obviously, the SOM correctly grouped together these ten cycles into one cluster.

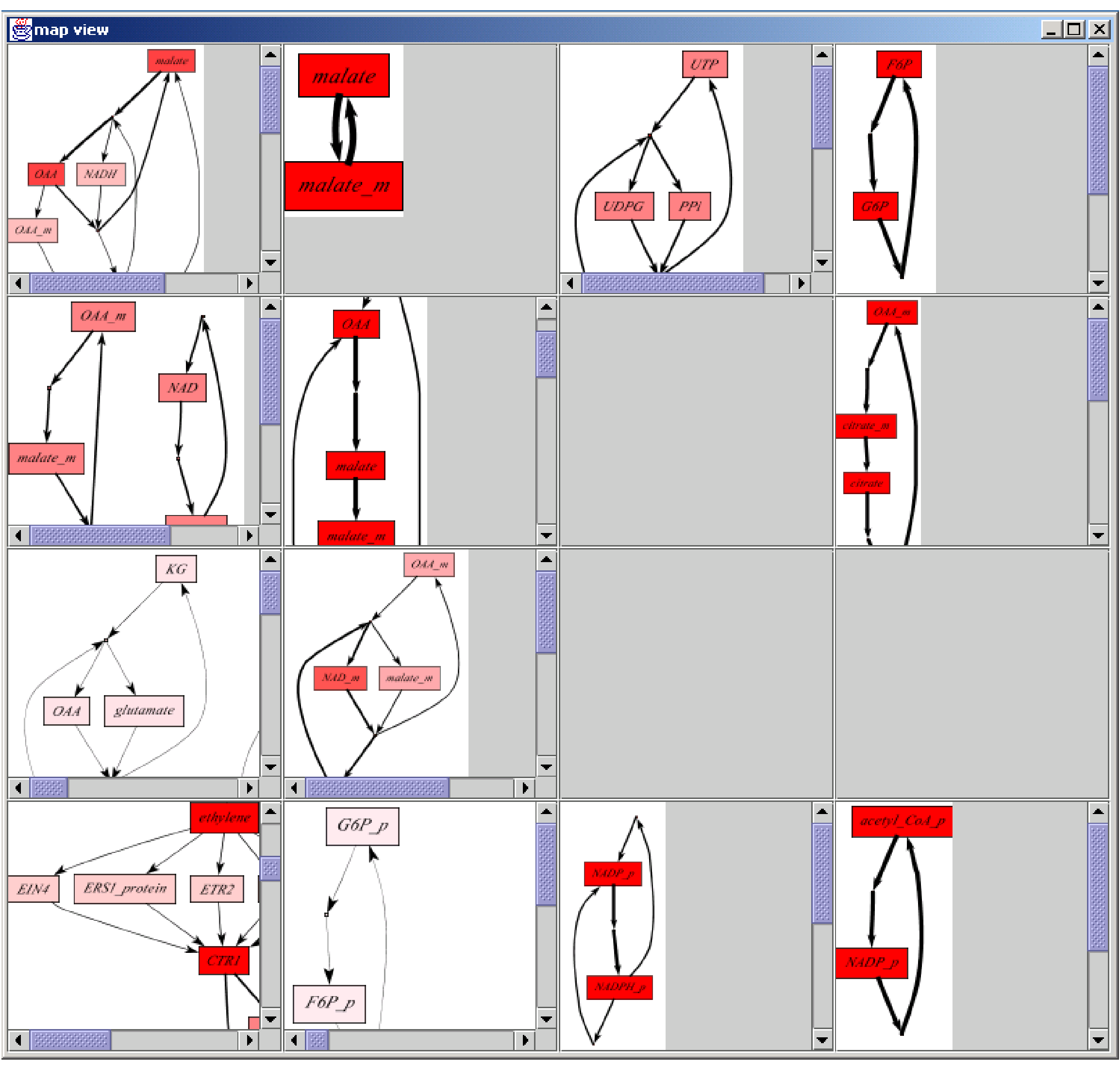

Figure 14. Map view showing results of the SOM algorithm. Each graph shows the model of the corresponding map unit, which is the generalized median of the cycles assigned to that map unit.

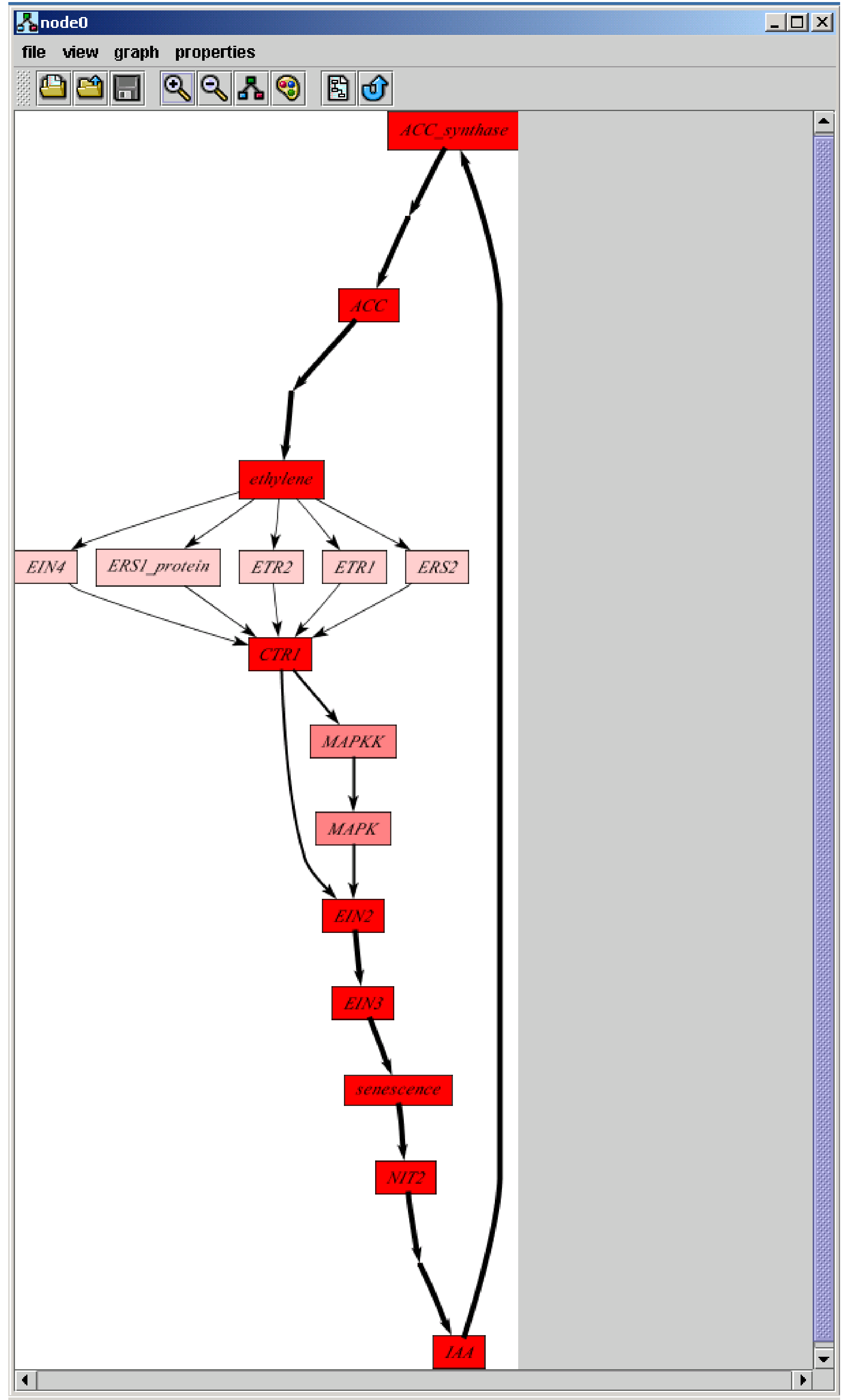

Figure 15. Full view of the bottom-left map unit, showing ten cycles clustered together.

9 FCModeler and R

In work

10 References

10.1 Publications relating to FCModeler

Cox, Z. (February 28, 2002). JSOMap: a Java-based Self-Organizing Map package. http://jsomap.sourceforge.net. Date accessed: June 18, 2002.

Dickerson, J. A., D. Berleant, et al. (to be published in 2002). Creating and Modeling Metabolic and Regulatory Networks Using Text Mining and Fuzzy Expert Systems. Computational Biology and Genome Informatics. C. H. Wu, P. Wang and J. T. L. Wang, World Scientific.

Dickerson, J. A., D. Berleant, et al. (2001). Creating Metabolic Network Models using Text Mining and Expert Knowledge. Atlantic Symposium on Molecular Biology and Genome Information Systems and Technology (CBGIST 2001), Durham, North Carolina.

Dickerson, J. A., Z. Cox, et al. (2001). Creating Metabolic and Regulatory Network Models using Fuzzy Cognitive Maps. North American Fuzzy Information Processing Conference (NAFIPS), Vancouver, B.C.

10.2 Open Source Code Used in FCModeler

Dot graph layout routine from the *Graphviz* package of AT&T (http://www.research.att.comsw/tools/graphviz).

Diva framework graph nodes and edges (http://www.gigascale.org/diva/)

Java SDK 1.3.x (http://java.sun.com/j2se/1.3/)

Tulip GEM Routine for graph layout (http://www.tulip-software.org)